Seminar 2: Intervaluator
Funkcijsko programiranje 2018/19

V tem seminarju boste implementirali jezik Intervaluator, "evalvator intervalov".

Vaš seminar bo ocenjen samodejno, zato je pomembno, da svoj izdelek pripravite natančno v skladu s podanimi navodili. Vstopna točka vašega programa naj bo funkcija `(iv exp env)`, kjer je `exp` Intervaluatorjev izraz in `env` okolje, v katerem bo izraz ovrednoten.

Intervaluatorjevi konstrukti naj bodo definirani z Racketovim konstruktom `struct`. Uporabite jih za definicije konstruktov, opisanih v sledečih odstavkih.

1 Podatkovni tipi

- **Številske vrednosti** `(const n)`**:** če je `n` številska vrednost v Racketu, potem je `(const n)` številska vrednost v Intervaluatorju.
- **Logične vrednosti** `(bool b)`**:** če je `b` logična vrednost v Racketu, potem je `(bool b)` logična vrednost v Intervaluatorju.
- **Intervali** `(interval a b)`**:** če sta `a` in `b` številski vrednosti v Racketu, potem je `(interval a b)` interval v Intervaluatorju s podanimi mejami (vključno z njimi). Interval je v teh navodilih predstavljen z zapisom $[a, b]$, kjer je $a \leq b$.
- **Pari** `(pair e1 e2)`**:** če sta `e1` in `e1` izraza v Intervaluatorju, potem je `(pair e1 e2)` par Intervaluatorjevih vrednosti `v1` in `v2`, ki sta rezultat vrednotenja izrazov `e1` in `e2`. Par je v teh navodilih predstavljen z zapisom (e_1, e_2).
- **Ničelna vrednost** `(nil)`**:** ta izraz predstavlja ničelno vrednost v Intervaluatorju (podobno kot `null` v Racketu). Pozor: `(nil)` je izraz, toda `nil` (brez oklepajev) ni.

2 Operacije in krmiljenje

- **Vejitev** `(if-then-else b e1 e2)`**:** ta izraz predstavlja vejitev v Intervaluatorju. Če se `b` ovrednoti v `(bool #t)`, potem je vrednost vejitve enaka vrednosti izraza `e1`, v nasprotnem primeru pa je enaka vrednosti izraza `e2`. Le en izmed izrazov `e1` in `e2` naj bo ovrednoten.
- **Poizvedbe tipov** `(is-const? e)`**,** `(is-bool? e)`**,** `(is-interval? e)`**,** `(is-nil? e)`**:** če je `e` izraz v Intervaluatorju, potem so `(is-const? e)`, `(is-bool? e)`, `(is-interval? e)` in `(is-nil? e)` izrazi v Intervaluatorju, ki se ovrednotijo v logične vrednosti `(bool #t)`, če je vrednost izraza `e` tipa `const`, `bool`, `interval` oz. `nil`, in `(bool #f)` v nasprotnem primeru.
- **Negacija** `(negate e)`**:** če je `e` izraz v Intervaluatorju, potem je `(negate e)` negacija vrednosti izraza `e`. Negacija logične vrednosti je nasprotna logična vrednost, negacija številske vrednosti v je številska vrednost $-v$, negacija intervala $[a, b]$ pa je interval $[-b, -a]$.

- **Seštevanje** `(add e1 e2)`: če sta `e1` in `e2` izraza v Intervaluatorju, potem je `(add e1 e2)` vsota vrednosti `v1` in `v2` izrazov `e1` in `e2`. Če sta `v1` in `v2` številski vrednosti, potem je vrednost izraza številska vrednost vsote `v1` in `v2`. Če sta `v1` in `v2` intervala $[a, b]$ in $[c, d]$, potem je vrednost izraza interval $[a + c, b + d]$. Če je ena izmed vrednosti `v1` in `v2` številska vrednost v in druga interval $[a, b]$, potem je vrednost izraza interval $[a + v, b + v]$.

- **Množenje** `(multiply e1 e2)`: če sta `e1` in `e2` izraza v Intervaluatorju, potem je `(multiply e1 e2)` zmnožek vrednosti `v1` in `v2` izrazov `e1` in `e2`. Če sta `v1` in `v2` številski vrednosti, potem je vrednost izraza številska vrednost zmnožka `v1` in `v2`. Če sta `v1` in `v2` intervala $[a, b]$ in $[c, d]$, potem je vrednost izraza interval $[\min\{ac, ad, bc, bd\}, \max\{ac, ad, bc, bd\}]$.

- **Eksponentna funkcija** `(exponentiate e)`: če je `e` izraz v Intervaluatorju, potem je `(exponentiate e)` eksponentna funkcija vrednosti `v` izraza `e`. Če je `v` številska vrednost v, potem je vrednost izraza številska vrednost e^v. Če je `v` interval $[a, b]$, potem je vrednost izraza interval $[e^a, e^b]$.

- **Ekstrakcija** `(left e)`, `(right e)`: če je `e` izraz v Intervaluatorju, potem sta `(left e)` in `(right e)` ekstrakciji vrednosti `v` izraza `e`. Če je `v` par (a, b), potem je `(left e)` vrednost a in `(right e)` vrednost b. Če je `v` interval $[a, b]$, potem je `(left e)` številska vrednost a in `(right e)` številska vrednost b.

- **Primerjava** `(greater e1 e2)`: če sta `e1` in `e2` izraza v Intervaluatorju, potem je `(greater e1 e2)` izraz v Intervaluatorju, ki predstavlja primerjavo. Če sta `v1` in `v2` numerični vrednosti izrazov `e1` and `e2`, potem je vrednost izraza `(bool #t)`, če `v1 > v2` in `(bool #f)` v nasprotnem primeru. Če sta `v1` in `v2` intervalski vrednosti izrazov `e1` in `e2`, potem je vrednost izraza `(bool #t)`, če je širina `v1` strogo večja od širine `v2` in `(bool #f)` v nasprotnem primeru. Širina intervala $[a, b]$ je $b - a$.

- **Presek** `(intersect e1 e2)`: če sta `e1` in `e2` izraza v Intervaluatorju, ki se ovrednotita v intervala $[a, b]$ in $[c, d]$, potem je `(intersect e1 e2)` interval v Intervaluatorju, ki ustreza preseku $[a, b] \cap [c, d]$ ali `(nil)`, če je presek prazen.

3 Spremenljivke

Implementirajte spremenljivke in okolje za njihovo shranjevanje. Ko poženete interpreter, je okolje prazno. Med njegovim delovanjem so vrednosti shranjene in prebrane. Implementirajte okolje z Racketovo nespremenljivo razpršeno tabelo (angl. immutable hash map). Definirajte sledeče konstrukte za shranjevanje in branje vrednosti:

- **Lokalno okolje** `(with vars e)`: če je `vars` Racketova razpršena tabela imen spremenljivk, preslikanih v njihove izraze in `e` izraz v Intervaluatorju, potem je `(with vars e)` lokalno okolje v Intervaluatorju. Trenutno okolje se razširi s spremeljivkami v `vars`, tako da se ovrednoti njihove izraze, nato pa se ovrednoti še izraz `e` v razširjenem okolju.

- **Branje spremenljivk** `(valof s)`: če je `s` niz v Racketu, potem je `(valof s)` izraz v Intervaluatorju, ki prebere vrednost spremenljivke `s`. Če spremenljivka `s` ne obstaja v trenutnem okolju, potem vrne `(nil)`. Okolje lahko vsebuje več zasenčenih vrednosti iste spremenljivke, zato je vrednost izraza enaka najbolj nedavna vrednost spremenljivke, ki zasenči vse prejšnje.

4 Funkcije

Implementirajte funkcije, skripte in klice. Funkcije uporabljajo leksikalni doseg, skripte pa dinamičnega. Nobena izmed definicij ne spreminja trenutnega okolja (za to uporabite konstrukt `with`). Definirajte sledeče konstrukte za delo s funkcijami, skriptami in klici:

- **Funkcije** `(function name farg body)`: če je `name` niz v Racketu ali `#f`, `farg` Racketov seznam imen formalnih argumentov in `body` izraz v Intervaluatorju, potem je `(function name farg body)` definicija funkcije v Intervaluatorju. Njena vrednost je ovojnica `(closure env f)`, kjer je `f` funkcija in `env` okolje, v katerem je bila funkcija definirana (leksikalni doseg). Okolje mora vsebovati tudi funkcijo samo, saj je to nujno potrebno za rekurzijo. Konstrukt `closure` **ni** del sintakse Intervaluatorja, ampak le notranji mehanizem vašega interpreterja, ki omogoča uporabo leksikalnega dosega. Ovojnice naj bodo tudi **optimizirane**, torej iz njih odstranite zasenčene spremenljivke ter tiste, ki niso potrebne za ovrednotenje telesa funkcije.
- **Skripte** `(script name body)`: če je `name` niz v Racketu ali `#f` in `body` izraz v Intervaluatorju, potem je `(script name body)` definicija skripte v Intervaluatorju. Skripte nimajo argumentov, le telo. Branje spremenljivk deluje v trenutnem okolju, kjer se skripta izvaja (dinamični doseg).
- **Klici** `(call e arg)`: če je `e` izraz v Intervaluatorju, ki se ovrednoti v ovojnico ali skripto in `arg` Racketov seznam Intervaluatorjevih izrazov, potem je `(call e arg)` klic v Intervaluatorju. Če se `e` ovrednoti v ovojnico, vrednost klica enaka vrednosti telesa funkcije, ki se ovrednoti znotraj njene ovojnice, razširjene s pari imen formalnih argumentov `farg` in njihovih vrednosti `arg`. Če se `e` ovrednoti v skripto, potem je vrednost klica enaka vrednosti telesa skripte, ki se ovrednoti v trenutnem okolju. Ker skripte ne potrebujejo argumentov, naj bo vrednost `arg` enaka `(nil)`.

5 Makro sistem

Implementirajte funkcije v Racketu, ki bodo delovale kot makri v Intervaluatorju, olepšale sintakso in razširile množico konstruktov. Definirajte sledeče makre:

- **Odštevanje** `(subtract e1 e2)`: generira Intervaluatorjev izraz, ki predstavlja razliko `e1 - e2`.
- **Primerjava** `(lower e1 e2)`: generira Intervaluatorjev izraz, ki preveri, ali je `e1` manjši kot `e2`.
- **Enakost** `(equal e1 e2)`: generira Intervaluatorjev izraz, ki preveri, ali je `e1` enak `e2`.
- **Vsebovanost** `(encloses i1 i2)`: generira Intervaluatorjev izraz, ki preveri, ali interval `i1` vsebuje interval `i1`.

5.1 Higieničen makro sistem

Ta naloga je opcijska. Implementirajte svoj makro sistem, ki je higieničen (ne izvede naivne razširitve, ampak loči med spremenljivkami v makru in v programski kodi). Odločitev o tem, kako boste implementirali tak sistem, je v celoti prepuščena vam. Lahko uporabite poljubne konstrukte v Racketu. Ta naloga ne bo ocenjena samodejno, ampak jo boste razložili in zagovarjali posebej.

6 Oddaja in točkovanje

Izvorno kodo oddajte v eni datoteki, imenovani `<vpisna št.>-sem2.rkt`, na učilnico pred predpisanim rokom.

Točkovanje bo izvedeno samodejno, svoje rešitve pa boste morali tudi zagovarjati. Samodejno testiranje bo izvedeno s testnimi primeri, ki ne bodo znani vnaprej. Svojo kodo testirajte z lastnimi testnimi primeri.

6.1 Točkovanje

1. Podatkovni tipi — 10 točk
2. Operacije in krmiljenje — 15 točk
3. Spremenljivke — 15 točk
4. Funkcije
   (a) Ovojnice — 15 točk
   (b) Skripte — 10 točk
   (c) Rekurzija — 10 točk
   (d) Optimizacija ovojnic — 10 točk
5. Makro sistem — 10 točk
   (a) Zahtevani makri — 10 točk
   (b) Higieničen makro sistem — (+10 točk)
6. Slog kode in testi — 5 točk